Chapter 11
The HTM Learning Algorithm

According to the fail fast principle in Chap. 4, we need to learn from systems' abnormal behavior and downright failures to achieve anti-fragility to classes of negative events. The earlier we can detect problems, the smaller the negative consequences are and the faster we can start learning how to improve the systems. Since humans are not good at detecting anomalies, especially in streaming data from large cloud applications, a form of automatic anomaly detection is needed. There are many ways to detect anomalies, depending on which complex adaptive system we consider. For example, Internet banking solutions employ a rich set of heuristics to detect fraud [26]. This first chapter of Part IV introduces a general learning algorithm based on Hawkins's developing theory of how the brain learns, called hierarchical temporal memory (HTM) [27, 96]. The HTM learning algorithm, or just HTM, is used in the next chapter to detect anomalies in a system's behavior. HTM was earlier referred to as the cortical learning algorithm.

The underlying basis for the HTM learning algorithm is not easy to understand and the algorithm itself is still being developed. To grasp HTM's novelty and importance, the current chapter first discusses the approach to learning taken by traditional artificial intelligence (AI) research, as well as efforts to "train" artificial neural networks to realize particular input–output mappings defined by data "training sets." Second, the chapter outlines why HTM is an improvement over these earlier approaches. Finally, it provides a fairly detailed description of the HTM learning algorithm (with some algorithmic details left out to ease understanding).

While Hawkins' general theory on how the brain works is very interesting, this chapter only provides enough information to understand the major steps of the HTM learning algorithm. The reader wanting to know more about the theory behind HTM should study Hawkins' book *On Intelligence* written with Sandra Blakeslee [27]. More technical information on HTM is given in a white paper [96] by Numenta (http://numenta.com), which was set up to develop the algorithm for both commercial and scientific use. An open source project, called the Numenta platform for intelligent computing (NuPIC) (http://numenta.org), provides HTM program code and documentation. YouTube (see also http://numenta.com/learn) has a growing number

K.J. Hole, *Anti-fragile ICT Systems*, Simula SpringerBriefs on Computing 1,
DOI 10.1007/978-3-319-30070-2_11

of talks by Hawkins and others at Numenta on different aspects of HTM. Ryan Price [97] and Michael Galetzka [98] have studied the capabilities and performance of HTM. This chapter is mainly based on information provided by the above sources.

11.1 The Problem with Classical AI Research

Learning to recognize known patterns and predict future patterns remains a major challenge for AI, because any autonomous agent needs these abilities to operate successfully in a changing environment. Decades of classical AI research, as well as all the work carried in the 1980s and early 1990s to train artificial neural networks, have not been able to replicate human learning [99, 100, 101]. The main mistake, according to Hawkins [27], was not understanding how the human brain learns and, instead, treating the brain as a computer that could be programmed to produce intelligent behavior. The goal of classical AI was to develop algorithms that would first match and then later surpass human intelligence. Unfortunately, the programs developed were only good at the particular task for which they were designed. The programs did not have the ability to generalize or to show flexibility in the face of changing circumstances. Finally, there were significant unsolved problems on how to represent knowledge in computers.

The author of this book was among the many postdoctoral researchers in the early 1990s developing algorithms to train simple models of neural networks. After much work by many scientists, it became evident that, while these trainable neural network models could learn relatively small problem instances, they did not scale to handle large instances due to an exponential increase in training time and a limited ability to generalize to new circumstances. More recent deep learning algorithms for multilayer neural networks solve larger problem instances and have many interesting applications [102]. However, as for the earlier learning algorithms, deep learning still requires custom training in batch mode for specific tasks and does not continuously learn like the brain.

The limited success of classical AI and neural network research has made many scientists suspect that the brain does not run a large collection of specialized learning algorithms. Other scientists still believe that specialized learning algorithms are needed to achieve a high degree of intelligence. Only future research will show who is correct.

11.2 An Alternative Approach to Learning

Hawkins [27] believes the best way to understand how the brain learns is to use its biology as guidance while thinking about learning as an algorithmic problem with a solution implementable on computers, perhaps in the cloud, or, even better, in silicon. Not everybody agrees that the brain is an algorithmic machine (or Turing machine)

[103]. Some scientists and philosophers believe the brain to be a different type of machine based on quantum effects [104, 105, 106]. Here, we do not discuss whether the brain is a quantum computer, since HTM learning is purely algorithmic.

Hawkins has based HTM on a 1978 hypothesis by Vernon Mountcastle [107]. It states that the neocortex uses essentially the same learning algorithm to process the signals from all of the body's sensory organs. HTM is a general learning algorithm and a memory system storing invariant representations of physical structures and abstract concepts. While a traditional specialized AI learning algorithm must be programmed in great detail, HTM is self-learning. Furthermore, whereas artificial neural networks must be trained offline using particular training sets, HTM learns in real time as the data come in. Finally, unlike artificial neural networks, which require retraining when the world changes, HTM is able to forget old representations and learn new representations in real time.

HTM is based on the assumption that the world has structure and is therefore predictable. The world is not chaotic and not homogeneous but complex. The complex but structured behavior of the world allows HTM to learn by creating invariant representations of common patterns reported by the senses. The patterns occur in sequences, enabling HTM to predict future behavior based on earlier experienced behavior. Here, a sequence is a set of patterns that generally accompany each other but not always in a fixed order. The important point is that patterns of a sequence follow each other in time, although the order may vary. HTM is adaptable, allowing it to learn changes in the environment. Old memories are removed and new memories are formed. While HTM models the processing of sensory streams for human vision, touch, hearing, and language, it can also be exposed to non-human sensory input streams such as web traffic, data from cloud computing infrastructures, financial market data, and weather data [96].

11.3 The Brain's Neocortex

To understand how the HTM learning algorithm works, it is advantageous to first study its biological basis [27, 96]. HTM is modeled after the structure and operation of the neocortex, or just *cortex*, in the brain. The cortex is responsible for learning. It is a sheet of neural tissue approximately 1,000 cm^2 in area and 2.5 mm thick. The cortex looks like a dinner napkin wrapped around the older areas of the brain. The cortex contains at least 30 billion nerve cells, or *neurons*. It consists of six layers formed by variations in the density of cell bodies, cell types, and cell connections. There are five layers of cells and one non-cellular layer. During early development, the cortex divides itself into dozens of functional areas, or regions, based on experience and needs. The function of a region is determined by the information that flows into it. Note that a region comprises all six layers of the cortex. In the following, we consider three important aspects of the neocortex and its neurons.

11.3.1 Communication

Classical artificial neural networks model the function of a neuron as a weighted summation of inputs followed by a non-linear operation on the sum. We now know from neuroscience that cortical neurons carry out much more complex operations. An important reason why HTM improves on previous attempts to train neural networks is that HTM utilizes a radically different neuron model, heavily inspired by cortical neurons. The neurons in the cortex communicate with each other via electrical and chemical signals. The signals are the basis of memory and learning in the cortex. As depicted in Fig. 11.1, a typical neuron consists of the cell body, or soma, many dendrites, and a single axon. The branch-like dendrites receive incoming signals from other neurons and the axon and its terminal branches transmit outgoing signals to other neurons. Some axons are coated with myelin, a fatty substance that insulates the axon and increases the speed of communication. Signals pass between neurons at connections called synapses. Note from Fig. 11.1 that neurons do not touch. There is a microscopic gap, denoted the synaptic cleft (see inset), between the axon of one neuron and the dendrite of another.

The signaling occurs roughly as follows: When neuron *A* receives a chemical signal from another neuron, neuron *A* becomes electrically charged relative to the surrounding fluid outside its membrane. The electrical charge travels down the axon, away from *A*'s soma, until it reaches a synapse. Inside the synapse is a group of storage sites, denoted vesicles, containing chemicals manufactured by the soma. When the electrical charge arrives at the synapse, it causes these vesicles to fuse with the synapse's cell membrane, spilling molecules, called neurotransmitters, into the

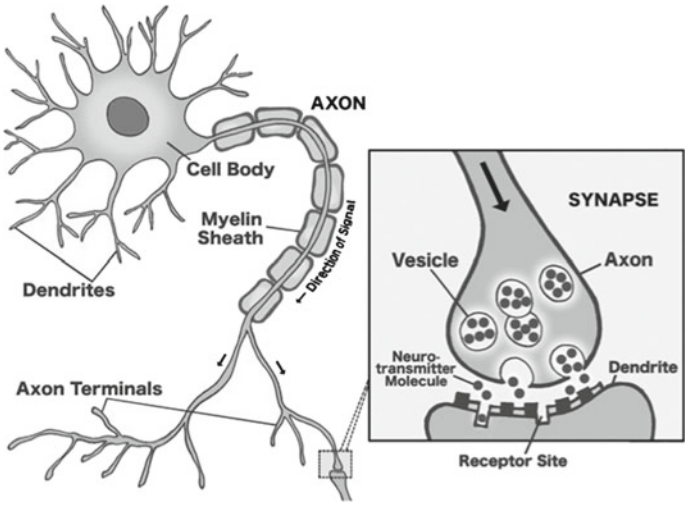

Fig. 11.1 Signal propagating down an axon to the dendrites of the next cells (figure from http://urbanchildinstitute.org/why-0-3/baby-and-brain)

synaptic cleft. The neurotransmitters move across the synaptic cleft to one of neuron *B*'s dendrites, where they bind with receptor sites in the dendrite's membrane. Neuron *B* develops an electrical charge, the charge travels down its axon, and the described process repeats itself.

While some cortical regions receive input directly from the body's sensory organs, other regions receive input only after it has passed through intermediate regions. The regions are connected via large bundles of axons or fibers. Information flows in parallel over these fibers at all times. The regions process a continuous stream of signals that create patterns in space and time inside the neocortex. The cortex does not experience the world directly; it only has access to patterns coming from the sensory organs. These patterns all have the same format inside the brain, allowing the cortex's different regions to use the same learning algorithm.

11.3.2 Memory

The cortex is not some kind of parallel computer that makes many computations on input patterns to create output patterns. Instead, the cortex rapidly retrieves outputs from its huge memory. All memories in the cortex are stored in the synaptic connections between neurons. While both the cortex and computers have memories, there are large differences: The cortex stores sequences of patterns, it recalls patterns auto-associatively, and it stores invariant patterns in hierarchies.

In more detail, the memory of the cortex automatically stores sequences of patterns. Memory recall almost always follows a path of association. Auto-associativity simply means that patterns are associated with themselves. An auto-associative memory can retrieve a complete pattern from a partial or noisy input sequence. This is true for both spatial and temporal patterns. The cortex is constantly completing patterns from noisy and partial inputs.

Regions are connected in hierarchies. Regions at a low level of a hierarchy store simple physical and abstract objects. These objects are combined into larger objects in higher regions. Simple objects can be a part of many hierarchies. Each region forms invariant representations of objects. The invariant representations allow the cortex to recognize faces and physical objects, although the light, viewing angle, and surroundings change all the time. The higher layers of the cortex combine information from the lower layers to understand multi-sensory inputs over time, for example, a film with both sound and moving images.

11.3.3 Predictions

The cortex combines the invariant representations with new inputs to make predictions about everything a human sees, feels, and hears. According to Hawkins [27], prediction is the primary function of the cortex and the basis for intelligence. We are

interested in the cortex's ability to predict, because an anomaly is detected when a prediction is violated. The reader should know that a great deal of information also flows downward in the hierarchies of the cortex. While these feedback connections are crucial to understanding how the brain creates behavior, they do not play an important role in the current version of HTM and will not be discussed here.

11.4 Overview of HTM

The HTM learning algorithm models how learning occurs in a single layer of the cortex. Input to the algorithm is a continuous stream of input patterns from some kind of system. HTM builds sparse, invariant representations of pattern sequences representing repeated structures in the input stream. The algorithm learns which patterns are likely to follow each other, thus learning to predict future patterns. When the HTM receives a novel pattern, it will try to match it to stored patterns. Because inputs never repeat in exactly the same way, invariance of the stored sequences is vital to the ability to recognize inputs.

Time plays a crucial role in HTM. Predictions can only be made on the basis of a sequence of earlier received patterns. Sometimes it is enough to know the previous pattern most recently received while at other times it is also necessary to know patterns received earlier. The ability to predict using variable-length sequences of patterns is due to the variable order memory of HTM. Note that HTM does not understand the meaning of patterns; it only knows what patterns are likely to follow particular observed patterns.

11.4.1 Sparse Distributed Representation

HTM generates internal sparse distributed representations (SDRs) of the input patterns. An SDR is given by a binary vector with a fixed number of bits. Different vector lengths are possible. A vector can contain 2,048 bits, only 2 % of which are ones, called active bits. The zero bits are the inactive bits. The individual bits in an SDR have semantic meaning, unlike, for example, the dense eight-bit ASCII code, where all bit patterns are used and the characters are assigned bit patterns randomly.

In an SDR, two inputs with similar semantic meaning must have similar binary vector representations, that is, they must have many equal bits when the vectors are compared position by position. This happens naturally for visually similar black and white pictures, while the binary representations of natural numbers with nearly the same values may not have a single bit in common, for example, $7 = 0111_2$ and $8 = 1000_2$. The SDR property is vital to HTM's ability to learn [27]. It is therefore often necessary to recode input data to HTM to ensure that vectors sharing active bits have similar semantic meaning. If the encoded input vectors are dense, then HTM creates a sparse representation.

Ahmad and Hawkins [108] have developed exact bounds on HTM's level of fault tolerance and robustness to noise. The bounds show that the use of SDRs makes it easy to construct HTM systems that are very robust to perturbations.

11.4.2 Proximal Dendrite Segments

HTM arranges artificial cells in 2,048 columns, with 32 cells in each column. Conceptually, the columns are arranged in a two-dimensional array, as illustrated in Fig. 11.2. Note that only a small part of the array is shown. Figure 11.3 illustrates how all the cells in a column share a single proximal dendrite segment receiving feed-forward input. Each column has potential connections to a random selection of the bits in an input vector to HTM. These bits are called the potential bits or the potential pool. The status of the connections is determined by the synapses in Fig. 11.3.

HTM uses the concept of *permanence* to change the connectedness of synapses. Permanence is a scalar value ranging from zero to one. It is assigned to a synapse to represent the degree of connectedness between the axon and the dendrite. A permanence value of zero represents a potential synapse that is not valid and has not progressed toward becoming a valid synapse. A permanence value above a threshold (typically 0.2) represents a synapse that has just connected but could easily be unconnected. A high permanence value, for example, 0.9, represents a synapse that is connected and cannot easily be unconnected. When a synapse's permanence is above a threshold, it is connected with weight one. Below the threshold, it is unconnected with weight zero. Note that there is no individual weighting of synaptic connections as in classical neural networks. Instead, HTM has the ability to create and remove these connections. According to Hawkins [27], HTM achieves a higher information

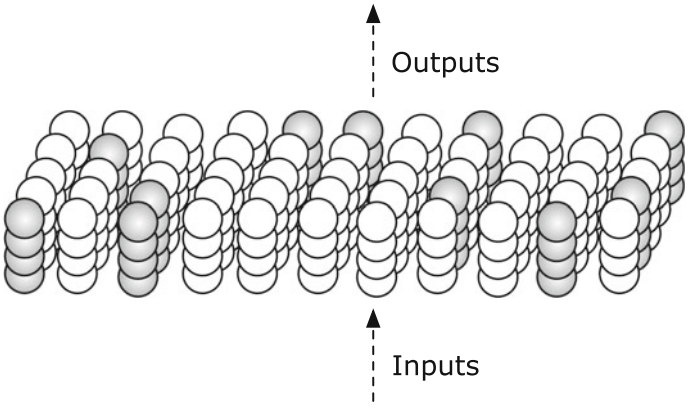

Fig. 11.2 HTM contains columns of cells with active cells shown in *gray*. When there is no prior state, all or none of the cells in a column are active

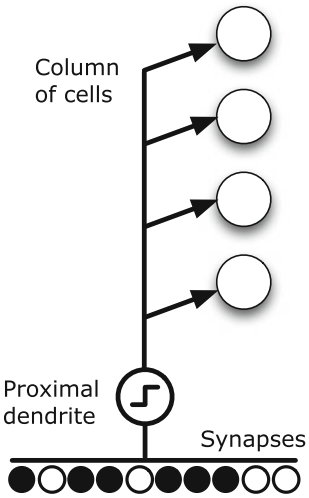

Fig. 11.3 The cells in a column share a proximal dendrite, with synapses represented by *small black circles*. A *solid circle* represents a valid synapse connection with a permanence value above the connection threshold and an *empty circle* represents a potential synapse connection with a permanence value below the connection threshold. Feed-forward input activates a column after a local inhibition step if enough valid synapses are connected to active input bits

storage capacity by forming and removing synaptic connections than changing the weights of permanent connections.

Each column determines its activation from the input vector by summing the input bits in positions with permanence larger than the threshold. The sum of the bits in these positions constitutes the *overlap score*. The higher the score, that is, the more active ones, the more overlap between the input and the pattern represented by the column. Columns with the greatest overlap (strongest activations) inhibit, or deactivate, columns with weaker activations. The inhibition function achieves a relatively constant percentage of (about 2 %, or 40) active columns, even when the number of input bits that are active varies significantly. The result is an SDR of the input encoded by which columns are active and inactive after inhibition.

11.4.3 Distal Dendrite Segments

In addition to the single proximal dendrite segment, a cell has about 130 distal dendrite segments, each with roughly 40 synapses. The distal segments receive lateral input from nearby cells. Figure 11.4 shows the distal dendrites and illustrates the cell's states. The set of potential synapses connects to a subset of other cells within a neighborhood defined by a "learning radius." A dendrite segment forms connections to cells that were active together at an earlier time, thus remembering the activation state of other cells in the neighborhood. If the same cellular activation pattern is encountered again by one of its segments, that is, the number of active synapses on any segment is above a threshold, the cell will enter a predictive state indicating that feed-forward input is expected to result in column activation soon.

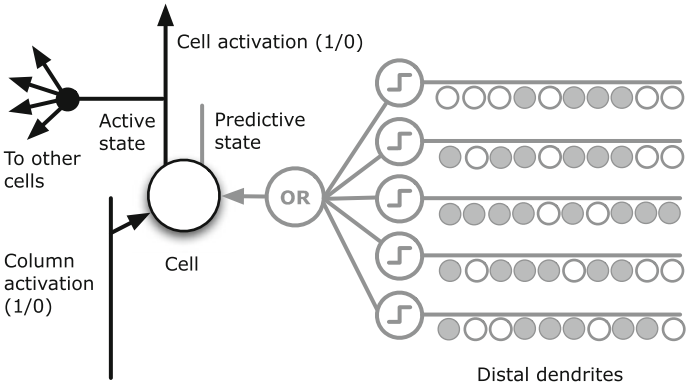

Fig. 11.4 Each distal dendrite segment of a cell has synapse connections to other cells within the neighborhood. A *solid gray circle* represents a valid synapse connection to another cell and an *empty circle* represents a potential synapse connection. The cell enters a predictive state if at least one of its dendrite segments is connected to enough active cells. A cell's binary-valued predictive state is not propagated. Column activation due to feed-forward input via the proximal dendrite is shown in *black* in the *bottom left*. The binary-valued active state is the feed-forward output of the cell and is also propagated to other cells via lateral connections depicted in the *upper left*

A cell is active due to feed-forward input via the proximal dendrite or lateral connections via the distal dendrite segments. The former is called the *active state* and the latter is called the *predictive state* (see Fig. 11.4). Only the feed-forward active state is connected to other cells in the region. The predictive state is internal to the cell and is not propagated. The complete output of HTM is a binary vector representing the active states of all cells.

11.5 The Three Steps of HTM

At each discrete time instance, HTM carries out three steps on the new input. The following descriptions of the steps, detailed in the next sections, are taken from [96]:

Step 1 Create an SDR of the input by activating whole columns.
Step 2 Place the input in context by selecting among cells in active columns.
Step 3 Predict future patterns from learned transitions between SDRs.

11.5.1 Make an SDR of the Input

The first step determines the active columns of cells in HTM (see Fig. 11.2). Each column is connected to a subset of the input bits via the synapses on a proximal dendrite. Subsets for different columns may overlap but they are not equal. Consequently,

different input patterns result in different levels of activation of the columns. The columns with the strongest activation inhibit columns with weaker activation. The size of the inhibition area around a column is adjustable and can span from very small to the entire region. The inhibition mechanism ensures a sparse representation of the input. If only a few input bits change, some columns will receive a few more or a few less active one inputs, but the set of active columns is not likely to change much. Therefore, similar input patterns will map to a relatively stable set of active columns.

HTM learns by forming and unforming connections between cells. Learning occurs by updating the permanence values of the synapses. Only the active columns increment the permanence value of synapses connected to active bits and decrement otherwise. Columns that do not become active for a long period do not learning anything. To not waste columns, the overlap scores of these columns are "boosted" to ensure that all columns partake in the learning of patterns.

11.5.2 Represent the Input in Context of Previous Inputs

Cells can be in one of three states. If a cell is active due to feed-forward input, then it is in the active state. If the cell is active due to lateral connections to other nearby cells, however, then it is in the predictive state; otherwise it is in the inactive state.

The second step converts the columnar representation of the input into a new representation that includes the past context. The new representation is formed by activating a subset of the cells within each column, typically only one cell per column. The rule used to activate cells is as follows: When a column becomes active, HTM checks all the cells in the column. If one or more cells in the column are already in the predictive state, only those cells become active. If no cells in the column are in the predictive state, then all the cells become active. The rule can be understood as follows: If an input pattern is expected, then HTM confirms that expectation by activating only the cells in the predictive state. If the input pattern is unexpected, then HTM activates all the cells in the column to signal that the input occurred unexpectedly.

By selecting different active cells in each active column, HTM can represent the exact same input differently in different contexts. Figure 11.5 illustrates how HTM can represent the sequence `AB` as part of two larger sequences `CABF` and `HABG`. The same columns have active cells in both cases but the active cells differ. If there is no prior state and therefore no context or prediction, all the cells in a column will become active when the column becomes active. This scenario occurs especially when HTM first starts processing input (see Fig. 11.2).

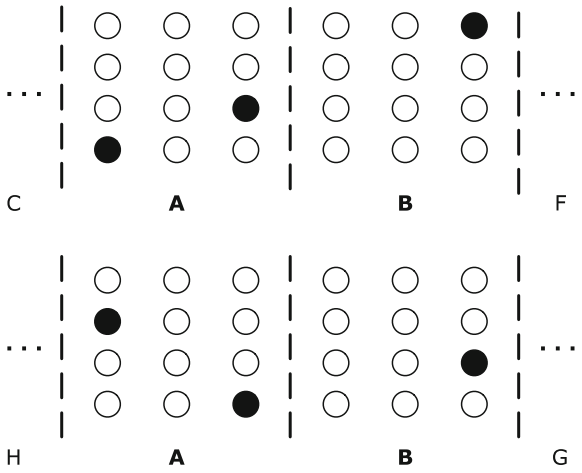

Fig. 11.5 The sequence AB is part of two larger sequences. The same active columns represent AB in both cases but the active cells differ because the larger sequences are different

11.5.3 Make Prediction from Current and Previous Inputs

The third and final step makes a prediction of likely new input. The prediction is based on the representation formed in the second step, which includes context from all previous input patterns. When HTM makes a prediction, all cells that are likely to become active due to future feed-forward input are changed to the predictive state. Because the representations are sparse, multiple predictions can be made at the same time instance. Together the cells in the predictive state represent HTM's prediction(s) for the next input.

The predictive state of any cell in HTM is determined by its distal segments. A segment connects to cells via synapses on distal dendrites. If enough of these cells are active, then the segment becomes active (see Fig. 11.4). A cell switches to the predictive state when it has at least one active segment. However, a cell that is already active from the second step does not switch to the predictive state. Learning occurs by adjusting the permanence values of the synapses on active segments at every time step. The permanence of a synapse is only updated when a predicted cell actually becomes active during the next time instance. The permanence of a synapse connecting to an active cell is increased while the permanence of a synapse to an inactive cell is decreased. (Note that the full update rules are significantly more complicated than those presented here. See [96] for a more detailed description of the rules.)

To apply the HTM learning algorithm to a particular data source, Numenta uses optimization techniques to choose optional HTM components, select parameter values, and determine which data fields to include (http://youtube.com/watch?v=xYPKjKQ4YZ0).

11.6 Discussion and Summary

Classical AI solutions are task specific and brittle; they can only do one major thing and they fail too easily. Hawkins [27] and Numenta [96] have developed and implemented a general learning algorithm, the HTM learning algorithm, that overcomes the weaknesses of the classical solutions [108]. HTM's general learning rules are to train on every input; if a pattern is repeated, then reinforce it; and if a pattern is not repeated, then forget it.

When a new input vector arrives, it leads to a sparse set of active cell columns. One or more of the cells in each column become active; these cells, in turn, cause other cells to enter a predictive state through learned lateral connections between cells in different columns. The cells activated by the lateral connections constitute a prediction of what is likely to happen next. When the next input vector arrives, it selects another sparse set of active columns. If a newly active column is unexpected—meaning that it was not predicted by any cells—it will activate all the cells in the column. If a newly active column has one or more predicted cells, only those cells will become active. The output vector contains the feed-forward output of all cells.

While the current HTM realization has hundreds of millions of synapses (300 million using the numbers in this chapter), the brain has trillions of synapses, making it clear that the HTM implementation is only simulating a tiny part of the brain. In the future, it should be possible to connect HTMs together in hierarchies to obtain more brain-like simulations. At the time of this writing, Hawkins and Numenta are working to introduce motor control into HTM and refining the functionality according to the behavior of the cortex. In the next chapter, we consider how to use HTM to detect anomalies in cloud-based systems.